

STATIC ANALYSIS BY INCREMENTAL COMPUTATION IN GO PROGRAMMING

K. Nakamura

*College of Science and Engineering, Tokyo Denki University
Hatoyama-machi, Saitama-ken, 350-0394 Japan.*

nakamura@k.dendai.ac.jp

Abstract Computer-Go programs have high computational costs for static analysis, even though most intersections of the board remain unchanged after one move. Therefore, we introduced the method of incremental computation as an essential feature in Go programming. This paper explores how incremental computation is applied to the static analysis in Go programs, and describes two types of analysis and pattern recognition. One type is *determination* in cases where the territories of groups are almost determined. This includes (1) the methods of determining the life and death of a group by numerical features and (2) the method of finding the numbers of regions enclosed by the groups based on Euler's formula. The other type is *estimation* of groups of stones and territories by analysing the influence of stones using an "electric charge model" in cases where the density of stones is rather low. In the analysis, operations on sets of intersections are used for mathematical descriptions when applying incremental computation as well as definitions of the notions on the Go board.

Keywords: incremental computation, Euler's formula, life and death, potential distribution, electric charge model

1. Introduction

The strength of computer-Go programs is generally considered as a beginners' level despite all efforts by many researchers. Many Go players in Japan estimate the current best Go programs as playing at around 4 or 5 *kyu* in amateur rating, although the Japan Go Association recently certified some Go programs as one *dan*. This is stronger than 5 *kyu*; the difference is 5 handicap stones. The progress in playing strength is considered rather slow compared to that of computer Shogi. The latter game is also considered very difficult, but apparently the Shogi programs are steadily improving. We assume that investigating the theoretical and mathematical foundations of the game as well as applying the results in practical Go programming are significant for computer Go.

It is widely accepted that an efficient static analysis is essential to improve the playing strength of computer-Go programs. However, the costs of such an analysis are much higher than those of chess and Shogi. The static analysis needs to be repeated not only at every move, but also at every step in the search tree.

In this paper, we explore how the incremental computation can be applied to static analysis. We discuss two types of static analysis and pattern recognition in computer Go: *determination* and *estimation*. The first type, *determination*, contains the analysis of cases where the territories of the groups have been almost determined. This includes (1) the methods of determining the life and death of a group by the numerical features and (2) the method of finding the numbers of regions enclosed by the groups based on Euler's formula. The other type, *estimation*, deals with the estimation of groups of stones and territories on the board when the density of stones is rather low by analysing the influence of stones using an electric charge model.

The aim of the static analysis is to obtain the *phase* of the board, which is a collection of overall aspects of the board configuration, such as territories of black and white stones, influence of stones, and life and death of the groups. In most cases, the change in board configurations is restricted to one intersection except for capturing, which seldom occurs. The largest part of the phase usually remains unchanged for one move, although there are cases where the phase changes vastly by one move. By using incremental computation for obtaining the phase of the board, we can restrict the evaluation process to the parts changed without repeating the same process for any unchanged part of the configuration. Since the game of Go requires high computational costs for the static analysis, incremental computation is especially effective for computer Go.

In most previous publications on static analysis in computer Go, the main subject dealt with determining the life and death of groups of stones. Those works include: the theoretical study of static life (Benson, 1976); determining the life and death of groups by some local features including perimeters of the empty regions (Chen and Chen, 1999) and by tactical analysis and eye values (Fotland, 2002); and static analysis by position evaluation (Müller, 2002). The application of combinatorial game theory to *yose* problems (Berlekamp and Wolfe, 1994) is another theoretical result. Nakamura (2000, 2001) presented basic approaches to the life-and-death problem, which included estimating the number of eyes based on Euler's formula for connected planar graphs and analysing capturing races by semeai graphs.

There are few papers that discuss the method of incremental computation in computer Go so far. Most Go-playing programs seem to have some mechanism for incremental computation. Klinger and Mechner (1996) and Bouzy (1997) describe some methods for incremental updating of data in Go programs. These

two publications contain elements of the basics of incremental computation since they take into account the knowledge maintenance and backtracking.

Since the early program by Zobrist (1969), most Go programs, including INDIGO (Bouzy, 1995), GO INTELLECT (Chen, 1989), HANDTALK (Chen, 2002), EXPLORER (Müller, 2002), and JIMMY 5.0 (Yan and Hsu, 2001) employ mechanisms for evaluating the influence of stones and determining territories. An important feature of our electric charge model is the computation of the potential distribution which is based on incremental computation. Another feature is that some aspects of Go boards can be described in detail by potential distributions.

This paper is organized as follows. In Section 2, we describe operations on the set of intersections on the board, which are used for representing features of pattern analysis as well as mathematical descriptions of incremental computation. Section 3 describes methods of recognizing blocks and groups based on the set operations, and discusses a method of identifying the life and death of a group enclosing a region by the numerical features of the regions defined by the set operations. Section 4 shows an improved method of estimating the number of regions enclosed by the groups based on Euler’s formula for planar graphs. Section 5 outlines another approach of static analysis for recognizing groups and finding the influence of stones based on the electric charge model and on incremental computation.

2. Set Operations and Incremental Computation

In this section, we define several constants and some operations on the sets of intersections. We show the relation of the operations with incremental computation. Our intention is not to use the sets of intersections and the operations directly for the analysis, but to define basic notions on Go boards and to use incremental computation only for the parts that changed in every move.

2.1 Operations on Sets of Intersections

The *Board* is the set $\mathcal{B} = \{(i, j) \mid 1 \leq i, j \leq N\}$ of *intersections*. In the standard rule N is 19. A configuration is represented by two disjoint sets $B \subseteq \mathcal{B}$ and $W \subseteq \mathcal{B}$ of intersections occupied by black and white stones, respectively. The intersections in B or W are called black or white stones, respectively. The other elements of Board, $\mathcal{B} - B - W$, are empty intersections. An intersection (i, j) is *adjacent to* an intersection (m, n) , if and only if $|i - m| + |j - n| = 1$. An intersection (i, j) is *adjacent to* a set S of intersections, if and only if $(i, j) \notin S$ and there is $(m, n) \in S$ such that (i, j) is adjacent to (m, n) .

The board \mathcal{B} and the empty set \emptyset are constants. Another constant is *Edge* \square , defined by

$$\square \triangleq \{(i, j) \mid i = 1, i = N, j = 1 \text{ or } j = N\}.$$

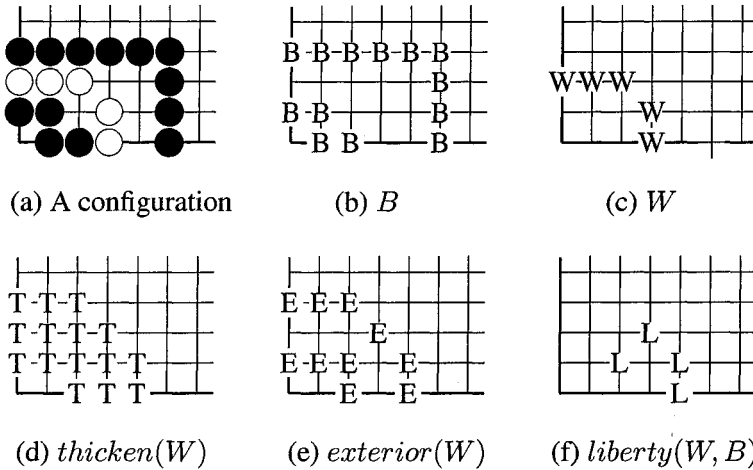


Figure 1. An example of a configuration and the results of extended operations.

We have three types of operations: Boolean, shift, and extended operations. The Boolean operations include union \cup , intersection \cap and set difference $-$. There are four shift operations. The operation Shift Left \overleftarrow{A} is defined by $\overleftarrow{A} \triangleq \{(i-1, j) \mid (i, j) \in A, i \geq 2\}$. The value of \overleftarrow{A} is the set of intersections which are shifted left from the intersections in A . The intersections on the left edge in A are eliminated. Other shift operations are: Shift Right \overrightarrow{A} , Shift Down $A \downarrow$ and Shift Up $A \uparrow$; they are defined analogously.

For a set X , it holds that $|X|$ is the number of elements in S . The following extended operations are used for representing features of enclosed regions in Subsection 3.3.

$$\begin{aligned} exterior(X) &\triangleq \{(i, j) \mid (i, j) \text{ is adjacent to } X\} \\ thicken(X) &\triangleq X \cup exterior(X) \\ \#adjacent(X) &\triangleq |X \cap \overrightarrow{X}| + |X \cap X \downarrow| \end{aligned}$$

Some examples of these operations are shown in Figure 1. We represent a configuration (Figure 1(a)) by sets B (Figure 1(b)) and W (Figure 1(c)) of black and white stones, respectively. The value of $\#adjacent(B)$ is 11, and that of $\#adjacent(W)$ is 3.

2.2 Operations and Incremental Computation

Let Y be any set of stones of the same colour, and A be a set of one stone of the same colour, such that $Y \cap A = \emptyset$. Incremental computation of an operation Op for $Y \cup A$ means finding the result $Op(Y \cup A)$ from the value

$Op(Y)$ and the operations on the neighbour intersections of A . The costs of incremental computation are generally lower than those of a full computation, since the change caused by adding a stone in A is restricted to the neighbour intersections of this stone. The results of incremental computation for the basic operations are as follows.

$$\begin{aligned} X \cup (Y \cup A) &= (X \cup Y) \cup A \\ X \cap (Y \cup A) &= (X \cap Y) \cup (X \cap A) \\ X - (Y \cup A) &= (X - Y) - A \\ (Y \cup A) - X &= (Y - X) \cup (A - X) \\ \overrightarrow{Y \cup A} &= \overrightarrow{Y} \cup \overrightarrow{A} \end{aligned}$$

Incremental computation for other shift operations is defined analogously. We note that the rightmost terms in the equations represent the changes. We also note that $A - X = \emptyset$, if $A \subset X$, and otherwise $A - X = A$. The results of the method for the extended operations are shown below, with $|S|$ being the number of elements in a set S .

$$\begin{aligned} thicken(Y \cup A) &= thicken(Y) \cup thicken(A) \\ exterior(Y \cup A) &= thicken(Y) \cup thicken(A) - (Y \cup A) \\ &= (exterior(Y) - A) \cup (exterior(A) - Y) \\ \#adjacent(Y \cup A) \\ &= |(Y \cup A) \cap (\overrightarrow{Y} \cup \overrightarrow{A})| + |(Y \cup A) \cap (Y \downarrow \cup A \downarrow)| \\ &= \#adjacent(Y) + |Y \cap exterior(A)| \end{aligned}$$

3. Static Analysis Based on Set Operations

In this section, we define blocks and groups, and discuss a method of determining the life and death of a group that depends on the shape of the enclosed region and on the positions of the opponent stones in the region. We implemented and tested most of the methods in Sections 3 and 4 in Prolog.

3.1 Blocks and Group

A *connected* set of intersections is defined by the following recursive rules.

- 1 A set of one intersection is connected.
- 2 For any set of T of intersections, if a subset $S \subseteq T$ is connected, then $thicken(S) \cap T$ is connected.

We represent a board configuration by sets B and W of black and white stones. The set E of empty intersections is given by $E = \mathcal{B} - B - W$. A *black block* is a connected set $B_X \subseteq B$ such that $thicken(B_X) \cap B = B_X$. *White*

blocks are defined analogously. An *empty region* is a connected set $E_X \subseteq E$ such that $\text{thicken}(E_X) \cap E = E_X$.

A *liberty*, or *dame*, of a black (or white) block B_X (W_X) is an empty intersection in the exterior of the block. Hence we have

$$\text{liberty}(B_X, W) \triangleq \text{exterior}(B_X) \cap E = \text{exterior}(B_X) - W.$$

We note that every block has non-empty liberties, since any block without the liberty is dead and removed from the board.

The configuration in Figure 1 (a) contains two black blocks, two white blocks and three small empty regions. The inner black block has two liberties. The two black blocks enclose the region of five white stones and five empty intersections.

A *group* is an important notion that is defined to be either a block or a union of blocks of the same colour such that the blocks are “dynamically” connected, i.e., the opponent cannot cut, or separate, the blocks. Although some groups, such as blocks connected by *kosumi* (diagonal) relations, can be recognized by static analysis, precise recognition needs dynamic analysis, as discussed in Nakamura (2002). We call the group in this narrow sense the *linked group*, which is a set of stones connected by adjacent-to or *kosumi* relations. In Section 5 it is shown that most of the groups in the broad sense are recognized by static analysis based on the electric charge model.

A group is alive, if the opponent player cannot capture it, and dead otherwise. Practically, a group is alive, if it has two eyes (i.e., small enclosed regions), it can be changed to form two eyes or a *seki*, or the group side wins the capturing race relating this group. There is a case where the life and death depends on a *ko* in the group.

3.2 Life and Death of Groups Enclosing Regions

Following Berlekamp and Wolfe (1994) and Chen and Chen (1999), we represent the types of enclosed regions related to the life and death of groups by pairs $\{\alpha|\beta\}$ of symbols, where α represents the state, if the group side moves next, and β , if the opponent moves next. The symbol of the state is either L , O , S , or K . Symbol L denotes that the group is alive in the sense that the enclosed region can form two eyes, whereas S denotes that the group enclosing the region can form a *seki*. Although the group is alive in the both cases, we distinguish S from L , because the opponent group in the region is also alive in the *seki*. Symbol O denotes that the region cannot be two eyes but only one eye. Symbol K denotes that the region can be changed to have a *ko* such that it can have two eyes, if the group side wins the *ko*, and one eye otherwise. Possible combinations in this section are $\{L|L\}$, $\{L|S\}$, $\{L|O\}$, $\{S|O\}$, $\{O|O\}$, $\{L|K\}$, and $\{K|O\}$. In some cases, life and death depends on the outer liberties of the group as well as the features of the region. Note that $\{L|L\}$ corresponds to 2.0 eyes, $\{L|O\}$ 1.5 eyes, and $\{O|O\}$ 1.0 eye in other

Feature	Definition
size of region R	$ R $
perimeter of R	$ exterior(R) $
num. of adjacent-to relations in X	$\#adjacent(X)$
max. neighbours in X	$max_neighbour(X)$
num. of opponent stones in R	$ B \cap R $
max. liberties of one stone in $B \cap R$	$max_liberties(B \cap R)$
total num. of opponent stones in R	$exterior(R \cap B)$
num. of intersections in R on the edge	$ R \cap \square $
outer liberties of the group	$ liberty(W) - E \cap liberty(W) $

Table 1. Features for determining the life and death of a group enclosing a region.

publications (Chen and Chen, 1999; Fotland, 2002). Since we discuss only the states of closed regions enclosed by groups and exclude the case where the region contains an opponent group with two eyes, we do not use the symbols for the states of half eyes or empty eyes.

Table 1 shows the list of features used for determining the life and death of the groups. In this table, R denotes the region, i.e., the set of intersections enclosed by a group, E the set of empty intersections, and B the set of opponent stones. We assume that the white group encloses the region in the figures. This table contains two features defined as follows.

$$max_neighbour(X) \triangleq \max_{p \in X} |X \cap exterior(\{p\})|$$

$$max_liberties(X) \triangleq \max_{p \in E} |E \cap exterior(\{p\})|$$

We tested this set of features for various patterns of the enclosed regions, and found that the features are effective to identify the types of life and death of the regions with the size of five to eight including those in the corners and those containing opponent stones.

Chen and Chen (1999) have shown that the life and death of a group enclosing an empty region R can be determined by the features, the perimeters of R , and the existence of square $\begin{smallmatrix} \perp \\ \perp \end{smallmatrix}$, which is given by $\#adjacent(R) - |R| + 1 \geq 1$ in our terminology. Another possible set of features for this recognition is $|R|$, $\#adjacent(R)$, and $max_neighbour(R)$.

Figure 2 shows typical empty regions in the corner with $\{L|K\}$ or $\{L|S\}$, if the groups have a few outer liberties. For example, the bent four in the corner (a) is in $\{L|L\}$, if the white group has two or more outer liberties, and $\{L|K\}$ otherwise. White can choose $\{L|K\}$ or $\{L|S\}$ in (e), if the outer liberties are zero or one. These patterns can be identified by the features, the number of intersections on the edge, the perimeters of R , and the number of squares, $\#adjacent(R) - |R| + 1$.

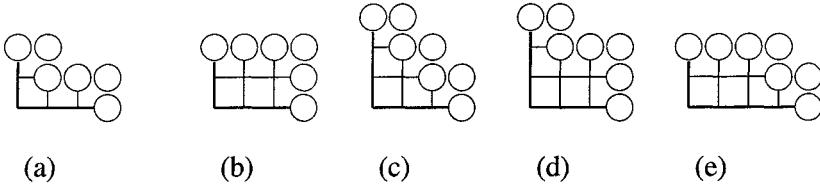
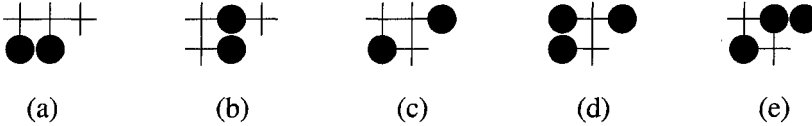


Figure 2. Patterns of groups with $\{L|K\}$ and/or $\{L|S\}$ when the white groups have a few outer liberties.



Pattern in Figure 3	(a)	(b)	(c)	(d)	(e)
num. of opponent stones	2	2	2	2	3
$\#adjacent(B \cap R)$	1	1	0	0	1
total liberties of $B \cap R$	2	3	3	3	2
max. liberties of one stone	1	2	2	2	1
outer liberties	-	-	≥ 1	0	-
Life and Death	$L O$	$O O$	$L L$	$L S$	$L L$
					$S S$

Figure 3. Regions with prisoners in size 5 enclosed by white groups.

Figure 3 shows examples of patterns of the enclosed regions containing two or three opponent stones (prisoners). We can identify each of the patterns by the five features in the table. Note that the groups enclosing the patterns (c) and (e) are alive by squashing (*oshitsubushi*), if the group has one or more outer liberties, otherwise the group is in *seki*.

A characteristic of our method is that the recognition is based on numerical features of the regions and groups, to which incremental computation can be applied. The method does not use pattern matching as used in many Go programs, which we consider inefficient and inappropriate for incremental computation.

The method shown in this section is only applicable to the groups enclosing closed regions. To analyse patterns with incompletely closed regions, or patterns with half eyes or open eyes, several methods have been proposed such as those by eye values and eye regions in Chen and Chen (1999) and Fotland (2002) and by position evaluation in Chen (2002). We are working on extending our methodology so that it can be applied to incompletely closed regions or loosely connected groups, e.g., those connected by *bamboo joints*.

4. Finding the Number of Enclosed Regions Based on Euler’s Formula

The regions enclosed by groups are important for deciding the life and death of the group, since the eyes are small enclosed regions and a group enclosing a region can be alive as discussed in the previous section. Nakamura (2000, 2002) proposed a method of using a formula to find the number regions enclosed by the groups. In this section, we show an improved method of finding the number of enclosed regions based on the method of incremental computation. The term “group” in this section refers to the linked group.



For any connected planar graph, the number N of regions enclosed by edges, or minimal loops, is given by Euler’s formula $N = n - k + 1$, where n and k are the numbers of edges and vertices, respectively. This formula has been applied in computer graphics to find the number of enclosed open regions in digital figures, which are represented by bit arrays (Gray, 1971). Euler’s formula is also applied in finding “holes” in the game Lines of Action (LoA) (Winands, Uiterwijk, and Van den Herik, 2001).

4.1 Application of Euler’s Formula to Go


For the application of Euler’s formula to graphs to find the number of enclosed regions in Go, we consider each stone in a group as a vertex, and each “link” between the stones as an edge. The *link* is either the “adjacent-to” relation or the diagonal relation of two stones in the group.

$$\#link(G) \triangleq \#adjacent(G) + |G \cap \vec{G}\downarrow| + |G \cap \overleftarrow{G}\downarrow|$$

It is remarked that we assume that every stone in a group is connected to at least an other stone in the group by the link.

The group may contain *closed loops* of stones composed of three stones and three links, e.g.,  and . To find the number of enclosed regions (or the number of open loops), the number of the closed loops $\#closed_loop(G)$ should be subtracted from the number of the minimal loops. The number of regions enclosed by the group G , is given by

$$\#empty_region(G) \triangleq \#link(G) - |G| - \#closed_loop(G) + 1.$$

A group may contain a closed loop of the form , which contains two diagonal links. In this case, only one of the diagonal links is valid, since Euler’s formula applies only to planar graphs. For example, the black group in Figure 4 has 16 links including 8 diagonal links, 12 stones and 4 closed loop. The number of enclosed regions is calculated as $16 - 12 - 4 + 1 = 1$. When the intersection A is occupied by a black stone, the numbers of links, stones and

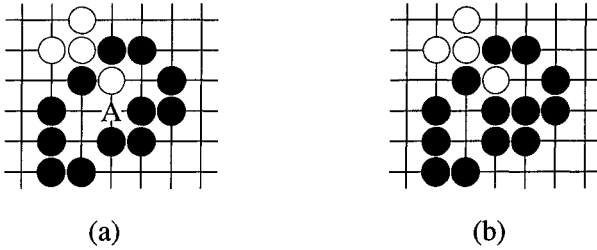


Figure 4. Black groups enclosing one region (a) and two regions (b).

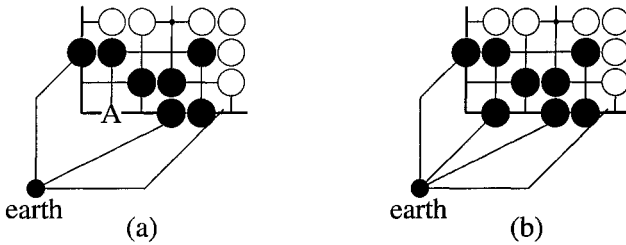


Figure 5. A group in the corner enclosing one region (a) and two regions (b).

closed loops increase by 3, 1 and 1, respectively, and the number of enclosed regions changes to $19 - 13 - 5 + 1 = 2$.

To apply this method to the groups in the peripherals and corners, we consider that there are links between stones on the edge of the board and a special virtual stone called *earth* as shown in Figure 5. To find the number of the virtual links, we first assign the set of stones on the edge $G \cap \square$ to a variable X . The number of virtual links is $|X|$, and the number of closed loops with the earth is $|X \cap X \downarrow| + |X \cap \vec{X}|$. We say that the group G is *earthed*, if $X \neq \emptyset$. Since the group in Figure 5 (a) has 11 links including 3 virtual links, 8 stones including earth and 3 closed loops, the number of open loops is $N = 11 - 8 - 3 + 1 = 1$. After placing a black stone at A , the number changes to $N = 13 - 9 - 3 + 1 = 2$.

4.2 Incremental Computation

For an effective incremental computation of the number of enclosed regions, we consider the change in number caused by placing a stone on an empty intersection p close to a black group G , i.e., there is a stone $q \in G$ such that there is a link between p and q . For the intersection $p = (i, j)$ not on the edge, let $C(p)$ be the circular sequence of eight neighbour states,

$$S_{i,j+1}, S_{i+1,j+i}, S_{i+1,j}, S_{i+1,j-1}, S_{i,j-1}, S_{i-1,j-1}, S_{i-1,j}, S_{i-1,j+1},$$

where each state $S_{x,y}$ is empty, or a black or white stone around the intersection p . The change in the number of regions caused by placing a black stone at p

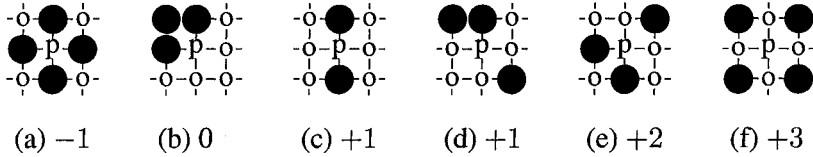


Figure 6. Typical patterns of neighbours and changes in the numbers of enclosed regions.

equals $E(p) - 1$, where $E(p)$ is the number of the consecutive subsequences in $C(p)$ satisfying the following conditions.

- 1 Each state is either empty or a white (opponent) stone.
- 2 Each subsequence contains one or more elements in $exterior(\{p\})$.

The fact that the change equals $E(p) - 1$ is derived from $E(p) = n' - 1 - L'$, where n' is the change in the number of links and L' is the change in the number of the closed loops caused by adding the stone. Figure 6 shows typical state patterns of neighbours and the increments of the number of regions enclosed by a black group. The symbol o denotes either an empty or a white stone. Note that for the intersection A in Figure 4 (a), the change is one, since $E(A) = 2$. Note also that the intersections p with $E(p) \geq 2$ are considered the vital points.

For the intersection p on the bottom edge, the number $E(p)$ is defined as the number of consecutive subsequence in the sequence,

$$S_{i-1,j}, S_{i-1,j+1}, S_{i,j+1}, S_{i+1,j+i}, S_{i+1,j}.$$

The number $E(p)$ is similarly defined for other edges with different directions, for the corners, and for the white stones. The change in the number of regions is $E(p) - 1$, if the group is earthed, and left and right neighbour intersections are empty. Otherwise, the change in the number of regions is $E(p) - 2$. Note that since a stone is placed on the edge in this case, the group changes in being earthed. Figure 7 shows typical patterns of neighbours and their changes in the number of regions. The pattern (c) represents that the change is one, if the group is earthed, and zero otherwise. Patterns (e) and (f) represent two cases in the corner intersection. Since the point A in Figure 5 (a) matches the pattern (c) and the group is earthed, the change in the number of regions is one.

4.3 Problems Related to Incremental Computation

The method shown in this section only provides the number of enclosed regions, but no information on the position of the regions, which are necessary for the analysis as performed in Section 3. A practical method for determining the position is using the potential distribution to be described in Section 5.

Moreover, the enclosed regions counted by the method might include false eyes. A false eye occurs, when two blocks are connected by two diagonal links. Hence, a region with one empty intersection enclosed by two blocks is

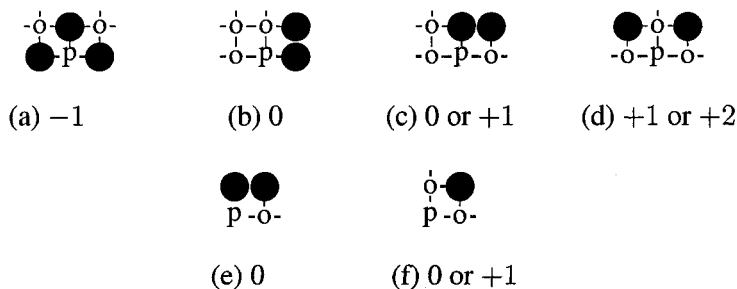


Figure 7. Typical patterns of neighbours on the edge (a - d) and the corner (e, f).

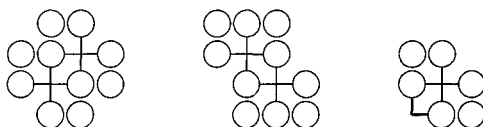


Figure 8. Groups consisting of two blocks connected by diagonal links.

not a false eye, if the blocks are connected by three or more diagonal links and virtual links (Figure 8). We can calculate the number of the connecting links by subtracting the total number of diagonal links in the two blocks from the number of diagonal links and virtual links in the group. Note that this rule is also effective for determining the life and death of the *dragon with two heads*, i.e., a group with two blocks connected by two false eyes. Although the condition for unconditional life by Benson (1976) covers these groups, our rule is simpler and appropriate for incremental computation.

Although most other enclosed regions can form one or two eyes as discussed in Section 3, there is the case that a large enclosed region containing an opponent group might form no eye and/or a *seki*. This problem can be solved by analysing capturing races (Nakamura, 2001).

5. Static Analysis by Electric Charge Model

This section outlines how incremental computation is used in estimating groups and territories based on the *electric charge model*. For a board configuration, the *potential* of each intersection is defined as follows.

- 1 Each stone distributes potential values $1/d$ to intersections around this stone, where d is the Manhattan distance between the stone and the intersection. The potential of every intersection is the sum of the potential values given by all the stones nearby. The potential given by black stones and that by white stones are separately calculated.
- 2 Stones close to the edges or the corners have their mirror images as shown in Figure 9. Therefore, the intersections near the edges or the corners have higher potential than intersections in the centre of the board.

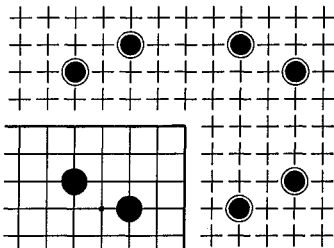


Figure 9. An example of mirror images.

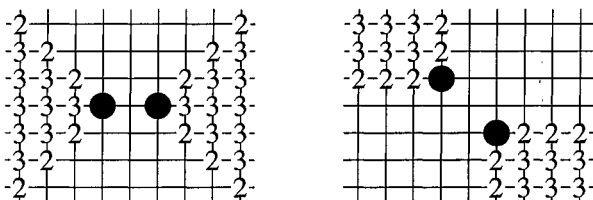


Figure 10. Values added to distances in shadows.

- 3 A potential value of an intersection given by a stone is reduced, if the intersection is in the *shadow* of another stone. The distance d is increased by the value shown in Figure 10 in the calculation of the potential value.

We use two different types of potential distributions. One is the potential distribution reflecting only the shadows of the stones of the same colour and the other is the potential distribution reflecting the shadows of all stones. In Subsection 5.3 the latter type of distribution is used for recognizing groups. The first type is intended to be used for estimating the strength of the groups, although the use is not shown in this paper.

Figure 11 shows examples of potential distributions. Because of the mirror images, the intersections in the corner have higher potentials as shown in Figure 11(c). In general, the potential of an intersection represents the degree to what extent the intersection is surrounded by stones. The potential in an enclosed region is approximately 4 to 6 as shown in Figure 11(b), and independent of the size and the shape of the region or of the stones of the opponent.

5.1 Incremental Computation of Potentials

Because of the linear, additive nature of the potential, incremental computation is generally simple, although mutual interactions of the shadows make the computation more complex for the configurations with many stones. We employ the following approximation method for computing a potential distribution.

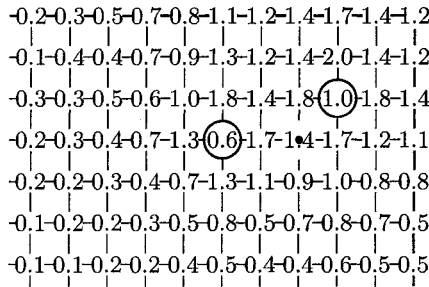
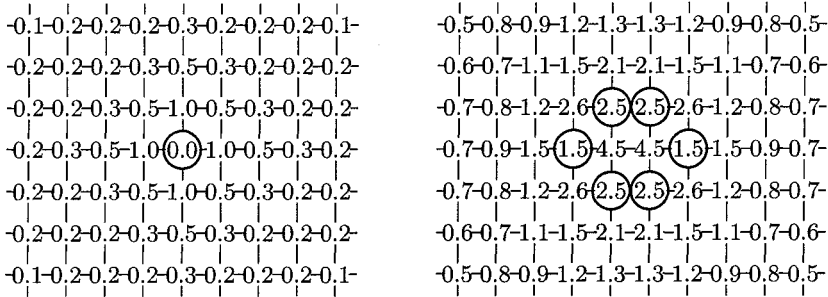


Figure 11. Examples of potential distribution.

- 1 We restrict the area to which the potential values from a stone are distributed to the set D of intersections in such a way that the distance from the stone to the other intersections is fewer than 8. The stones have their mirror images, only if the distance between the stones and the edge is fewer than 5.
- 2 Whenever a stone is placed on an intersection:
 - (a) the potential of every intersection in the area D is increased by the value $1/d$, where d is the distance between the stone and the intersection; and
 - (b) for each stone in the area D , the decrements by the effect of shadows are subtracted from the potentials of intersections in the two symmetric shadows of the two stones.

Note that the potential at an intersection obtained by the computation method above is slightly different from the one given by the definition in the previous subsection, if the intersection is in double shadows. The errors by the approximation, however, are negligible.

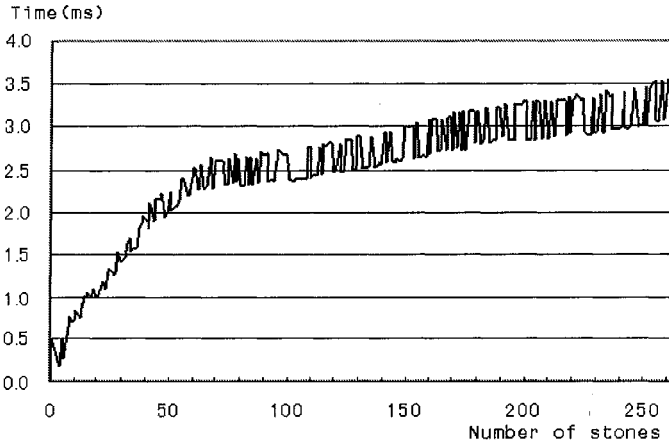


Figure 12. Computation time for potential distributions by computing the difference .

Figure 12 shows a graph of computation time of the potential distributions for each move. For the experiment, we used a Pentium III processor with 1G Hz clock running a program written in C++. The computation results include four potential distributions, i.e., those for two types of distributions for both black and white stones. Although the time increases with the number of stones while the number is fewer than approximately 50, the time is almost constant otherwise.

5.2 Recognition of Groups and Territories by Potential Distributions

Let $B(i, j)$ be the potential of an intersection (i, j) given by black stones, and $W(i, j)$ by white stones. We use the potential distribution that has the effects of shadows by both black and white stones. The procedure for recognizing black groups is as follows.

- 1 First, select *group points* from a given configuration by the following rules.
 - (a) The intersection occupied by a black stone is a black group point.
 - (b) An empty intersection is a black group point, if $B(i, j) \geq v_1$ and $B(i, j) - W(i, j) \geq v_2$, where v_1 and v_2 are parameters.

The white group points are selected similarly. Based on many experiments, we determined the parameters as $v_1 = 1.0, v_2 = 0.55$.

- 2 Determine connected sets of group points of the same colour (Note that this process is similar to that for determining blocks). The set of stones in each of the connected sets is a group. The connected set of group points represents the influence range, or the territory, of the group.

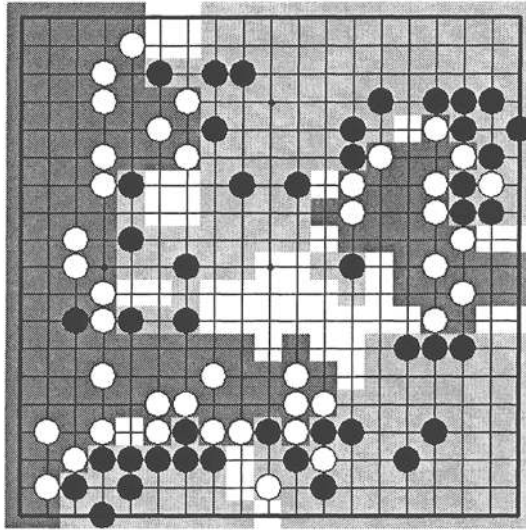


Figure 13. An example of groups and their territories derived from potential distributions.

- 3 In some cases, groups connected by diagonal, or *kosumi*, relations are not recognized only by the rules above. Hence, it is necessary to unify these groups into one by finding stones with this relation.

By testing this method for various configurations in games by professional players, we found that incremental computation can correctly recognize most groups for the wide range of configurations with more than 20 stones. Figure 13 shows an example of groups in a game (Black: C. Chou and White: M. Takemiya, 1994) evaluated by incremental computation. The dark gray areas represent the white territories, and light gray areas the black territories.

It is generally not difficult to compute the difference in finding the groups, since a group usually expands in each move and the changes of the group points are restricted to intersections near to the point of the move. There is, however, the case that a group is cut and separated by erroneous move(s) or *ko* threats. This case will be investigated in future research.

5.3 Comparison with Other Approaches

Most Go programs employ some methods of evaluating influence of stones and/or finding territories, including the potential distribution (Zobrist, 1969), the 5/21 algorithm (Bouzy, 1995), and the heuristic territory evaluation by Müller (2002). A feature of our approach is that each stone distributes the potential of $1/d$ to the neighbour intersections for the distance d . This methodology is common to those in several Go programs including HANDTALK (Chen, 2002), GO-INTELLECT (Chen, 1989), and JIMMY 5.0 (Yan and Hsu, 2001) in the

sense that each stone distributes, or radiates, some inference values to neighbour intersections. In contrast, these programs employ different methods for calculating the values except *HANDTALK*, in which the distribution of values is similar to our distribution led by $1/d$. For example, *GO-INTELLECT* uses the exponential function ($\exp(-d)$) instead of $1/d$.

A unique feature of our method in addition to incremental computation is that the influences of black stones and white stones are separately calculated. This is different from most other programs, in which the influence values by black (or white) stones are subtracted from those by white (black) stones to form a single distribution of influence values.

Another unique feature of our method is that it uses the mirror images, the shadows, and four kinds of potential distributions to describe some aspects of Go boards in detail. By these features, the potential values of every intersection in the board represent how strong other black and white stones surround the intersection. Note that this property is based on the potential given by $1/d$ and the mirror images.

6. Concluding Remarks

In this paper, we discussed static analysis based on incremental computation to be used in the static analysis in Go programming. The main questions were: (1) how the incremental computation can be applied to the static analysis, (2) how much does the computation speed increase by incremental computation, and (3) what sort of analysis is suitable or unsuitable for incremental computation? We showed applications of our method to static analysis in Go programming, including:

- identifying the life and death of a group enclosing a region by numerical features, which are described by the operations on sets of intersections;
- finding the number of regions enclosed by a group based on Euler's Formula; and
- estimation of groups and territories by potential distributions based on the electric charge model.

The analysis methods are based on numerical features or values, and not on pattern matching. Most notions in the static analysis and incremental computation are mathematically defined by the operations on sets of intersections. We showed that incremental computation can be used for the operations in the analysis.

The author and his colleagues are implementing the methods described above in a Go-playing program in Prolog and C++. There is still some work to be done before we can satisfactorily answer the questions above. Future problems include:

- finding numerical features effective for identifying alive-and-dead patterns of loosely connected groups, especially those in the corners;

- developing a method of acquiring a broad class of alive-and-dead patterns and making a database efficiently; and
- developing faster algorithms for the analysis, especially for incremental computation of the potential distributions.

Acknowledgements

The author would like to thank the anonymous referees for their helpful comments. He also would like to thank Shuhei Kitoma, Tomomi Miyashita, Hiroyuki Otsuka, and Ayumi Kondo for their help in implementing the ideas and preparing the manuscript.

References

- Benson, D.B. (1976). Life in the Game of Go. *Information Sciences*, Vol. 10, pp. 17–29.
- Berlekamp, E. and Wolfe, D. (1994). *Mathematical Go – Chilling Gets the Last Point*. A. K. Peters, Ltd.
- Bouzy, B. (1995). *Modelisation Cognitive du Joueur de Go*. Ph.D. Thesis, Université Paris, Paris.
- Bouzy, B. (1997). Incremental Updating of Objects in Indigo. *Fourth Game Programming Workshop*, Hakone, Japan, pp. 179–188.
- Chen, K. (1989). Group Identification in Computer Go. *Heuristic Programming in Artificial Intelligence*, D. Levy and D. Beal (eds.), pp. 195–210. Ellis Horwood, Chichester, UK.
- Chen, K. and Chen, Z. (1999). Static Analysis of Life and Death in the Game of Go. *Information Sciences*, Vol. 121, pp. 113–134.
- Chen, Z. (2002). Semi-Empirical Quantitative Theory of Go – Part 1: Estimation of the Influence of a Wall. *ICGA Journal*, Vol. 25, No. 4, pp. 211–218.
- Fotland, D. (2002). Static Analysis in THE MANY FACES OF GO. *ICGA Journal*, Vol. 25, No. 4, pp. 203–210.
- Gray, S.B. (1971). Local Properties of Binary Images in Two Dimensions. *IEEE Transactions on Computers*, Vol. C-20, No. 5, pp. 551–561.
- Klinger, K. and Mechner, D. (1996). An Architecture for Computer Go, <http://www.cns.nyu.edu/~mechner/compgo/acg/>.
- Müller, M. (2002). Position Evaluation in Computer Go, *ICGA Journal* Vol. 25, No. 4, pp. 219–228.
- Nakamura, K. and Kitoma S. (2002). Analyzing Go Board Patterns Based on Numerical Features. *Journal of IPSJ*, Vol. 43, No. 10, pp. 3021–3029 (in Japanese).
- Nakamura, K. (2001). Analyzing Capturing Races and Seki Situations. *Advances in Computer Games 9*, H.J. van den Herik and H. Iida (eds.), pp. 295–311.
- Nakamura, K. (2000). Graph-Theoretic Analyses of Go Board Phases. *Games in AI Research* H.J. van den Herik and H. Iida (eds.), pp. 239–250.
- Sanechika, N. (1988). Methods in Go System GO SEDAI. ICOT TM-0618 (in Japanese).
- Winands, M.H.M., Uiterwijk, J.W.H.M., and van den Herik, H.J. (2001). The Quad Heuristic in Lines of Action. *ICGA Journal*, Vol. 24, No. 1, pp. 3–15.
- Yan, S.-J. and Hsu, S.-C. (2001). A Positional Judgment System for Computer Go. *Advances in Computer Games 9*, H.J. van den Herik and H. Iida (eds.), pp. 313–326.
- Zobrist, A.L. (1969). A Model of Visual Organization for the Game of Go. *Proceedings of the AFIPS Spring Joint Computer Conference*, Vol. 34, pp. 103–112.